Procesamiento del Lenguaje Natural, Revista nº 49 septiembre de 2012, pp 171-178 recibido 05-05-12 revisado 28-05-12 aceptado 04-06-12

Optimizing Planar and 2-Planar Parsers with MaltOptimizer

Optimizando los Parsers Planar y 2-Planar con MaltOptimizer

Miguel Ballesteros†, Carlos Gómez-Rodríguez‡, Joakim Nivre§
†Universidad Complutense de Madrid, Spain
‡Universidade da Coruña, Spain
§Uppsala University, Sweden
miballes@fdi.ucm.es, carlos.gomez@udc.es, joakim.nivre@lingfil.uu.se

Resumen: MaltOptimizer es una herramienta capaz de proporcionar una optimización para modelos generados mediante MaltParser. Los analizadores de dependencias actuales requieren una completa configuración para obtener resultados a la altura del estado del arte, y para ello es necesario un conocimiento especializado. Los analizadores Planar y 2-Planar son dos algoritmos diferentes y de reciente incorporación en MaltParser. En el presente artículo presentamos cómo estos dos analizadores pueden incluirse en MaltOptimizer comparándolos con el resto de familias de algoritmos incluidas en MaltParser, y cómo se puede definir una búsqueda y selección de atributos (o "features") usando el propio sistema para estos dos parsers. Los experimentos muestran que usando estos métodos podemos mejorar la precisión obtenida hasta un porcentaje absoluto del 8 por ciento (labeled attachment score) si lo comparamos con una configuración básica de estos 2 parsers.
Palabras clave: Análisis sintáctico de dependencias, MaltOptimizer, MaltParser, Planar y 2-Planar

Abstract: MaltOptimizer is a tool that is capable of finding an optimal configuration for MaltParser models, taking into account that nowadays dependency parsers require careful tuning in order to obtain state-of-the-art results, and this tuning is normally based on specialized knowledge. The Planar and 2-Planar parsers are two different parsing algorithms included in MaltParser. In the present paper, we show how these two parsers can be included in MaltOptimizer processes comparing them with the rest of MaltParser algorithm families, and how we can define a deep feature search and selection by using MaltOptimizer for these two algorithms. The experiments show that by using MaltOptimizer we can improve parsing accuracy for Planar and 2-Planar parsers by up to 8 percent absolute (labeled attachment score) compared to default settings.
Keywords: Dependency parsing, MaltOptimizer, MaltParser, Planar and 2-Planar

1. Introduction

Data-driven applications are very useful because as soon as we have new data, they can be applied to different domains just by training the new models. However, this training normally requires careful tuning in order to obtain a reliable outcome. MaltOptimizer[1] (Ballesteros and Nivre, 2012b; Ballesteros and Nivre, 2012a) automates[2] the search of an optimal configuration for models obtained with MaltParser (Nivre, Hall, and Nilsson, 2006), which is a typical example of a widely used data-driven system. This requires finding preliminary parameters, such as the handling of covered roots or multiple root labels, selecting the parsing algorithm that best suits the data, and finally, a backward and forward feature selection with the intention of making use of all the possible information included in the data format. MaltOptimizer is an expert system, since it was built by using previous experience in the optimization of MaltParser models during the last years.

MaltParser, which is a transition-based dependency parser with state-of-the-art performance for many languages, was one of the top parsers in the CoNLL Shared Tasks

[1] http://nil.fdi.ucm.es/maltoptimizer

[2] MaltOptimizer is fully automatic and it can be run in batch mode, but it also allows an interaction with the user between phases.

ISSN 1135-5948

on dependency parsing (Buchholz and Marsi, 2006; Nivre et al., 2007). It implements different families of transition-based parsing algorithms: (i) Nivre's algorithms (Nivre, 2003; Nivre, 2008), (ii) Covington's algorithms (Covington, 2001; Nivre, 2008), and (iii) Stack algorithms (Nivre, 2009; Nivre, Kuhlmann, and Hall, 2009). However, there is another family that was not handled by the initial version of MaltOptimizer: (iv) the Multiplanar parsers (Gómez-Rodríguez and Nivre, 2010), which include two algorithms: the Planar parser and the 2-Planar parser. These are linear-time algorithms that cover the sets of *multiplanar* dependency structures described by Yli-Jyrä (2003): while the Planar parser is limited to dependency graphs with no crossing arcs, which are a rather tight superset of projective dependency graphs, the 2-Planar algorithm supports the vast majority of phenomena present in natural language treebanks, and does so in linear time and with competitive accuracy (Gómez-Rodríguez and Nivre, 2010).[3] For these reasons, the Planar and 2-Planar algorithms have been used in several applications and studies in recent literature (Ott and Ziai, 2010; Krivanek and Meurers, 2011; Beuck, Köhn, and Menzel, 2011; Gómez-Rodríguez and Fernández-González, 2012a; Gómez-Rodríguez and Fernández-González, 2012b).

This is why we believe it necessary to include the Multiplanar parsers in MaltOptimizer, and in this paper we present how we addressed this problem. We present how the Multiplanar parsers can be included into the system decision trees and how we refined the feature selection for these parsers.

In the rest of the paper, we introduce transition-based dependency parsing and the Planar and 2-Planar parsers (Section 2). We describe MaltOptimizer in a deeper way by also citing some similar systems and the modifications that we included in order to host these parsers (Section 3). We report experiments, showing how MaltOptimizer finds suitable parameters and feature models when the Multiplanar parsers are selected (Section 4). Finally, we show conclusions and plans for future work (Section 5).

2. The Planar and 2-Planar Dependency Parsers

Given a sentence $w_1 \dots w_n$, the goal of a dependency parser is to assign it a *dependency graph*, which is a directed graph $G = (V, A)$ where $V = \{1, \dots, n\}$ and $A \subseteq V \times V$.[4] We will assume that such dependency analyses are required to satisfy the *acyclicity constraint* (i.e. that the graph cannot have cycles) and the *single-head constraint* (that a node cannot have more than one incoming arc).

Transition-based dependency parsers assign dependency analyses to natural language sentences by using non-deterministic state machines, called *transition systems*, whose actions (transitions) manipulate input words and build dependency relations between them. The choice of the particular sequence of actions to parse each given sentence is performed by scoring transitions using a model learned from training data, and using these scores to select a suitable transition sequence. In the particular case of MaltParser, SVM classifiers are used to learn the model, and the selection of a transition sequence is done by greedy deterministic search, which proceeds by choosing the highest-scoring transition at each parser state.

The Planar and 2-Planar parsers, introduced by Gómez-Rodríguez and Nivre (2010), are among the parsing algorithms implemented in MaltParser. These algorithms use the transition systems shown in Figure 1. We give here a brief description of how both parsers work, a more thorough explanation can be found in Gómez-Rodríguez and Nivre (2010).

The Planar parser, like other algorithms implemented in MaltParser, uses two data structures to manipulate input words: a *buffer*, which holds the words that have not yet been read, and a *stack*, containing words that have already been read but that we still may wish to connect to other words via dependency arcs. The SHIFT transition is used

[3] In theory, the Multiplanar family of parsers is an infinite hierarchy of parsers with increasing coverage (an m-Planar parser can be defined for any natural number m). However, only the parsers with $m = 1$ and $m = 2$ have been implemented in MaltParser because the 2-Planar parser already has a very large coverage of non-projective phenomena (Gómez-Rodríguez and Nivre, 2010), so the practical interest of m-Planar parsers with $m > 2$ is dubious.

[4] In practice, the arcs in the set A are labeled, but we ignore arc labels in this explanation for simplicity of presentation.

Planar transition system:

Initial/terminal configurations: $c_s(w_1 \dots w_n) = \langle [], [1 \dots n], \emptyset \rangle$, $C_f = \{\langle \Sigma, [], A \rangle \in C\}$

Transitions:		
	SHIFT	$\langle \Sigma, i\|B, A\rangle \Rightarrow \langle \Sigma\|i, B, A\rangle$
	REDUCE	$\langle \Sigma\|i, B, A\rangle \Rightarrow \langle \Sigma, B, A\rangle$
	LEFT-ARC	$\langle \Sigma\|i, j\|B, A\rangle \Rightarrow \langle \Sigma\|i, j\|B, A \cup \{(j,i)\}\rangle$ only if $\nexists k \mid (k,i) \in A$ (single-head) and $A \cup \{(j,i)\}$ is acyclic.
	RIGHT-ARC	$\langle \Sigma\|i, j\|B, A\rangle \Rightarrow \langle \Sigma\|i, j\|B, A \cup \{(i,j)\}\rangle$ only if $\nexists k \mid (k,j) \in A$ (single-head) and $A \cup \{(i,j)\}$ is acyclic.

2-Planar transition system:

Initial/terminal configurations: $c_s(w_1 \dots w_n) = \langle [], [], [1 \dots n], \emptyset \rangle$, $C_f = \{\langle \Sigma_0, \Sigma_1, [], A \rangle \in C\}$

Transitions:		
	SHIFT	$\langle \Sigma_0, \Sigma_1, i\|B, A\rangle \Rightarrow \langle \Sigma_0\|i, \Sigma_1\|i, B, A\rangle$
	REDUCE	$\langle \Sigma_0\|i, \Sigma_1, B, A\rangle \Rightarrow \langle \Sigma_0, \Sigma_1, B, A\rangle$
	LEFT-ARC	$\langle \Sigma_0\|i, \Sigma_1, j\|B, A\rangle \Rightarrow \langle \Sigma_0\|i, \Sigma_1, j\|B, A \cup \{(j,i)\}\rangle$ only if $\nexists k \mid (k,i) \in A$ (single-head) and $A \cup \{(j,i)\}$ is acyclic.
	RIGHT-ARC	$\langle \Sigma_0\|i, \Sigma_1, j\|B, A\rangle \Rightarrow \langle \Sigma_0\|i, \Sigma_1, j\|B, A \cup \{(i,j)\}\rangle$ only if $\nexists k \mid (k,j) \in A$ (single-head) and $A \cup \{(i,j)\}$ is acyclic.
	SWITCH	$\langle \Sigma_0, \Sigma_1, B, A\rangle \Rightarrow \langle \Sigma_1, \Sigma_0, B, A\rangle$

Figure 1: The planar and 2-planar transition systems. Note that we use the notation $\Sigma|i$ for a stack with i at the top and tail Σ, and $i|B$ for a buffer with i as its first word and tail B.

to read the first word from the buffer, moving it to the stack. The LEFT-ARC and RIGHT-ARC transitions create a leftward (rightward) dependency arc involving the topmost stack node and the first node remaining in the buffer, and can only be executed if this arc does not create a cycle or violate the single-head constraint when combined with the already created arcs. Finally, the REDUCE transition can be employed to remove a word from the stack when we do not need to create more arcs using it. This set of actions allows the Planar parser to build any dependency graph that is planar, i.e., not containing crossing arcs. Note that planarity is a very mild relaxation of the well-known notion of projectivity, meaning that the practical coverage of the planar parser is only very slightly larger than that of projective dependency parsers.

To expand this coverage further and obtain a parser that would be able to analyze in linear time the vast majority of dependency structures present in natural language treebanks, the Planar parser was extended by adding an extra stack, obtaining the 2-Planar parser. The 2-Planar parser is capable of building any dependency graph that can be divided into two subgraphs (planes) that are planar. To do so, it uses two stacks, one per plane, such that one of them is marked as the *active* stack at each given configuration, the other being the *inactive* stack. The SHIFT transition works analogously to that in the Planar parser but it moves the first word in the buffer to both stacks. The REDUCE, LEFT-ARC and RIGHT-ARC transitions also work like their Planar counterparts, but they involve only the active stack. Finally, an additional SWITCH transition makes the active stack inactive and vice versa.

The paper by Gómez-Rodríguez and Nivre (2010) shows that the 2-Planar parser can produce results that are on par with well-known state of the art dependency parsers, like the arc-eager pseudo-projective parser by Nivre et al. (2006).

3. *MaltOptimizer*

MaltOptimizer (Ballesteros and Nivre, 2012b; Ballesteros and Nivre, 2012a) implements a search of different parameters for MaltParser based mainly on the heuristics described by Nivre and Hall (2010) and previous experience acquired during the last years. MaltOptimizer takes a single input, which is a training set in CoNLL data format,[5] and returns suggestions of an optimal configuration

[5] http://ilk.uvt.nl/conll/#dataformat

for MaltParser models, providing a complete option file and a feature specification file.

MaltOptimizer also estimates the expected results by providing labeled attachment score results (LAS) (Buchholz and Marsi, 2006).[6] It only explores linear multiclass SVMs in LIBLINEAR (Fan et al., 2008), excluding LIBSVM (Chang and Lin, 2001) for efficiency reasons. It has been observed that the expected outcomes are similar between both libraries but LIBLINEAR takes much less running time. This fact makes the experiments shown in this paper even more interesting, because most of the feature models that have been obtained manually for Multiplanar parsers are based on LIBSVM (Gómez-Rodríguez and Nivre, 2010; Gómez-Rodríguez and Fernández-González, 2012a; Gómez-Rodríguez and Fernández-González, 2012b).

It is worth noting that both LIBSVM and LIBLINEAR provide outcomes in the same range of accuracy, or as stated in (Prudhvi Kosaraju and Kukkadapu, 2010) or (Gómez-Rodríguez and Fernández-González, 2012b), LIBLINEAR can even provide better results than LIBSVM.

There are some other systems with the same intention as MaltOptimizer, due to the importance of feature selection and parameter optimization in machine-learning based systems. However, in the NLP community, the set of such systems is very limited. We can find feature selection and parameter optimization systems, such as Kool, Zavrel and Daelemans (2000) and Daelemans et al. (2003). More recently, Nilsson and Nugues (2010) explored automatic feature selection specifically for MaltParser, but MaltOptimizer is the first system that implements a complete customized optimization process for this system and for a big set of algorithms.

MaltOptimizer is divided in three different phases: (i) data analysis, (ii) parsing algorithm selection and (iii) feature selection. In the following subsections we describe each of them and we show the modifications that we had to make in order to host the Multiplanar parsers in the MaltOptimizer processes.

[6] In the default settings, it provides LAS including punctuation symbols, but it can be configured excluding punctuation symbols and excluding the labeling returning unlabeled attachment scores (UAS).

3.1. Phase 1: Data Analysis

In the first phase, MaltOptimizer gathers some information that leads the optimization for the following phases. Some of the properties are crucial in order to select the best parsing algorithm, which makes them very important due to the aim of the present paper.

1. Size of the training set: number of words/sentences.
2. Existence of "covered roots" (arcs spanning tokens with HEAD = 0).
3. Frequency of labels used for tokens with HEAD = 0.
4. Percentage of non-projective arcs/trees.
5. Existence of non-empty feature values in the LEMMA and FEATS columns.
6. Identity (or not) of feature values in the CPOSTAG and POSTAG columns.

In order to host Multiplanar parsers, we did not change the behavior of the first phase at all. The first three properties are used to set basic parameters, such as the way of handling covered roots by the model; 4 is used in the choice of parsing algorithm (phase 2), and for Multiplanar parsers it is basically the same as for other algorithms, the modifications are shown in the Phase 2 description; 5 and 6 are relevant for feature selection experiments (phase 3).

3.2. Phase 2: Parsing Algorithm Selection

The original version of MaltOptimizer handled three different groups of transition-based parsing algorithms: (i) Nivre's algorithms (Nivre, 2003; Nivre, 2008), (ii) Covington's algorithms (Covington, 2001; Nivre, 2008), and (iii) Stack algorithms (Nivre, 2009; Nivre, Kuhlmann, and Hall, 2009). However, this initial release was not able to handle Multiplanar parsers (Gómez-Rodríguez and Nivre, 2010), which is basically the goal of this paper. The Multiplanar group of algorithms contains algorithms that can handle non-projective dependency trees (2-Planar or 2-Planar arc-eager parser), and a roughly projective[7] version (Planar or Planar arc-eager) that can be combined with

[7] As mentioned above, the Planar parser has coverage over the set of planar dependency graphs, which is a rather tight superset of the set of projective dependency graphs.

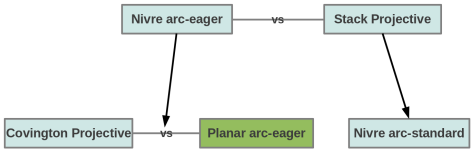

Figure 2: New decision tree for best projective algorithm in which Planar arc-eager is considered.

pseudo-projective parsing to recover non-projective dependencies in post-processing by using the pseudo-projective parsing approach (Nivre and Nilsson, 2005).

MaltOptimizer has two different decision trees, the first one is only used if the number of non-projective dependencies in the training set is small (or zero), and the second one is used if the number of non-projective trees is not zero. Therefore, for most of the languages both decision trees were explored in previous experiments (Ballesteros and Nivre, 2012b), with some exceptions, such as Chinese.[8]

We needed to locate the Multiplanar parsers in the decision trees in order to be able to select them as best parsers. Planar arc-eager has the same parsing order and a similar way of handling left and right attachments as Nivre arc-eager, this is why we locate it in the same branch in which the main parent is Nivre arc-eager, making a comparison with Covington projective. The new projective decision tree is shown in Figure 2.

When the training set contains a substantial amount of non-projective dependencies, the older version of MaltOptimizer tests the non-projective versions of Covington's algorithm and the Stack algorithm (including a lazy and an eager variant), and also the projective algorithms in combination with pseudo-projective parsing. We have 2-Planar arc-eager and Planar arc-eager, which is a (roughly) projective parsing algorithm, run in combination with pseudo-projective parsing. Therefore, according to parsing order the Multiplanar parsers are classified with Covington and arc-eager. The new non-projective decision tree is shown in Figure 3.

At the end of Phase 2, if 2-Planar arc-eager has been selected as best option, then MaltOptimizer tries with and without its

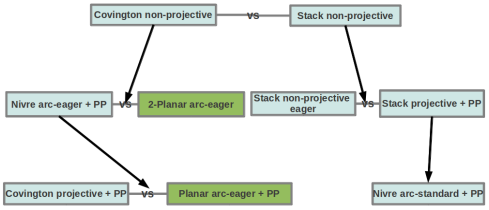

Figure 3: New decision tree for best non-projective algorithm (+PP for pseudo-projective parsing) in which Planar arc-eager and 2-Planar arc-eager are considered.

`-2pr` option, which can improve performance for some datasets by applying a REDUCE transition automatically after each SWITCH transition. There is also a `-prh` option, governing whether root arcs from the artificial root node 0 are constructed explicitly by the algorithm or left to be added automatically after parsing, and MaltOptimizer also explores it in order to find the best possible configuration.

3.3. Phase 3: Feature Selection

Once MaltOptimizer has gathered the preliminary parameters and the best parsing algorithm for the input data, it starts with the more challenging problem: the feature selection. MaltOptimizer tunes the feature model given all the parameters chosen so far, such as the parsing algorithm. It starts with default feature models for each parsing algorithm and then it tries with a greedy feature selection. In order to host Multiplanar parsers, we needed to modify the behavior in some cases. However, it is worth noting that the new processes are similar to the ones already implemented for Nivre arc-eager.

The features in MaltOptimizer can be explained dividing them into different feature windows: (i) part-of-speech, (ii) morphology, (iii) partially built dependency structure (dependency relation features), (iv) coarse grained part-of-speech, (v) extra morphological features such as gender or number (FEATS column) and (vi) lemma features.[9] For each window, it tries with backward selection experiments to ensure that all features in the default model for the given parsing algorithm are actually useful. After that, the system proceeds with forward selection experiments, trying potentially useful features

[8] To check statistics about non-projectivity in the training corpora, please visit http://ilk.uvt.nl/conll/paper_submission.html#table

[9] For an explanation of the different feature columns see (Buchholz and Marsi, 2006) or see http://ilk.uvt.nl/conll/#dataformat

one by one.

The major steps (and modifications to include Multiplanar parsers) of the feature selection experiments are the following:

1. Tune the window of POSTAG n-grams over the parser state.
   - It has been observed empirically (Gómez-Rodríguez and Nivre, 2010) that the planar algorithm tends to benefit from adding more features than in arc-eager, so for Multiplanar parsers we extend the search a bit over the buffer and the parser state.
   - For 2-Planar arc-eager we needed to distinguish between the active stack and the inactive stack, adding possible features involving the inactive stack when they are available (see Section 2). This fact was a challenge, because the inactive stack data structure is something new that is not available in any other parsing algorithm.
2. Tune the window of FORM features over the parser state. We added the same modifications as the ones added for POSTAG feature selection.
3. Tune DEPREL and POSTAG features over the partially built dependency tree.
4. Add POSTAG and FORM features over the input string.
5. Add CPOSTAG, FEATS, and LEMMA features if available. For 2-Planar arc-eager, MaltOptimizer tries with new features for the inactive stack when available.
6. Add conjunctions of POSTAG and FORM features.

Additionally, while in Nivre's arc-eager parser the first word in the buffer cannot be linked to any other, in Multiplanar parsers this can (and does) happen, so it makes sense to have features using the head word and dependency label associated with the first position of the buffer (Input[0]). This is a general modification that we included all over the process when Planar and 2-Planar are selected as best parsers.

4. Experiments

In this Section we present two different sets of experiments with corpora from the CoNLL-X Shared Task on multilingual dependency parsing (Buchholz and Marsi, 2006). We force the optimizer (by using the possible interaction between phases) to use Planar arc-eager and 2-Planar arc-eager as selected parsing algorithms,[10] in order to run a full feature selection for these two parsers and to observe how far we can go with the updated feature selection.

Table 1 shows the results for all the MaltOptimizer phases. Default and Phase 1 columns present the outcomes of the first phase in which Nivre arc-eager was selected as default parsing algorithm. Phase 2 columns, is divided in 2, in which we show the results of Planar and 2-Planar.[11] Finally, in Phase 3, we show the results of feature selection again for Planar and 2-Planar, interacting with the possibility that MaltOptimizer provides, stopping the process between phases. For phase 3, when the training corpora has non zero non-projective arcs/trees we run Planar arc-eager with pseudo-projective parsing, otherwise, we run it with default settings.

			Phase 2		Phase 3	
Language	**Default**	**Phase 1**	**Planar**	**2-Planar**	**Planar**	**2-Planar**
Arabic	63.02	63.03	62.81	63.42	65.53	64.94
Bulgarian	83.19	83.19	82.89	84.09	83.55	84.09
Chinese	84.14	84.14	81.66	82.79	83.63	83.54
Czech*	69.85	70.24	70.34	70.45	75.60	74.76
Danish	81.01	81.01	80.86	81.18	82.08	82.75
Dutch	74.77	74.77	76.55	76.81	76.55	81.41
German	82.36	82.36	81.34	82.28	84.64	84.84
Japanese	89.70	89.70	86.62	88.19	87.95	89.79
Portuguese	84.11	84.31	84.06	84.19	84.06	86.10
Slovene	66.08	66.52	65.94	66.43	69.72	70.13
Spanish	76.45	76.45	75.69	76.52	78.29	79.15
Swedish	83.34	83.34	82.38	82.83	83.67	83.78
Turkish	57.79	57.79	55.94	56.08	64.44	64.00

Table 1: Labeled attachment score per phase and with comparison to default settings for the training sets from the CoNLL-X shared task (Buchholz and Marsi, 2006). *The results for Czech were obtained using half of the training corpus due to a memory heap issue.

We can note that the performance improved substantially over all the selected corpora. However, for some data sets we got a much more substantial improvement, such as Turkish or Slovene, than in the other corpora.

[10] The Multiplanar parsers are not selected as best parsers for the corpora in which we carried out experiments with MaltOptimizer, due to the action of Stack non-projective and Stack projective parsing algorithms, which provide normally the highest results.

[11] For 2-Planar, we force MaltOptimizer to select between the 2-Planar options (`-2pr` and `-prh`) making it believe that 2-Planar is the best parsing algorithm for the data.

It is also worth noting that 2-Planar arc-eager normally provides better results, both in default settings (Phase 2) and in the optimized version (Phase 3), than Planar arc-eager. However, in some cases this is not the case, and the feature selection for Planar arc-eager reaches a higher attachment score.

5. Conclusions and Future Work

In this paper we have demonstrated that MaltOptimizer, which is an open-source system, can be updated with new parsing algorithms that are included (or will be) in MaltParser. This fact demonstrates that it is a very useful tool in order to get a reliable outcome when a user wants to use MaltParser for a new data set.

We have also demonstrated that the improvement is substantial, and we therefore suggest using MaltOptimizer when Planar and 2-Planar parsers are going to be used in comparison between parsers, so as to obtain reliable results.

As future work, we intend to add new feature selection algorithms, not only for Planar and 2-Planar parsers. Therefore, we are thinking on new procedures that select the best feature set possible and make them able to beat the results shown in previous publications of MaltParser feature selection and the present one. We could even consider a more complex version of MaltOptimizer, in which we run Phase 3 before Phase 2 for all the algorithms, or a subset of them, and then we could guarantee better results. However, for big data sets, it will take a lot of time to run several feature selection experiments.

Acknowledgments

MB has been funded by the Spanish Ministry of Education and Science (TIN2009-14659-C03-01 Project).
CGR has been partially funded by the Spanish Ministry of Economy and Competitiveness and FEDER (project TIN2010-18552-C03-02) and Xunta de Galicia (Rede Galega de Recursos Lingüísticos para unha Sociedade do Coñecemento).

References

Ballesteros, Miguel and Joakim Nivre. 2012a. MaltOptimizer: A System for MaltParser Optimization. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC)*.

Ballesteros, Miguel and Joakim Nivre. 2012b. MaltOptimizer: An Optimization Tool for MaltParser. In *Proceedings of the System Demonstration Session of the Thirteenth Conference of the European Chapter of the Association for Computational Linguistics (EACL)*.

Beuck, Niels, Arne Köhn, and Wolfgang Menzel. 2011. Incremental parsing and the evaluation of partial dependency analyses. In *Proceedings of the Int. Conference on Dependency Linguistics (Depling 2011)*.

Buchholz, Sabine and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 149–164.

Chang, Chih-Chung and Chih-Jen Lin, 2001. *LIBSVM: A Library for Support Vector Machines*. Software available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm.

Covington, Michael A. 2001. A fundamental algorithm for dependency parsing. In *Proceedings of the 39th Annual ACM Southeast Conference*, pages 95–102.

Daelemans, Walter, Véronique Hoste, Fien De Meulder, and Bart Naudts. 2003. Combined optimization of feature selection and algorithm parameters in machine learning of language. In Nada Lavrac, Dragan Gamberger, Hendrik Blockeel, and Ljupco Todorovski, editors, *Machine Learning: ECML 2003*, volume 2837 of *Lecture Notes in Computer Science*. Springer.

Fan, R.-E., K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. 2008. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874.

Gómez-Rodríguez, Carlos and Daniel Fernández-González. 2012a. Dependencias no dirigidas para el análisis basado en transiciones. *Procesamiento del Lenguaje Natural*, 48:43–50.

Gómez-Rodríguez, Carlos and Daniel Fernández-González. 2012b. Dependency parsing with undirected graphs.

In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 66–76, Avignon, France, April. Association for Computational Linguistics.

Gómez-Rodríguez, Carlos and Joakim Nivre. 2010. A transition-based parser for 2-planar dependency structures. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1492–1501.

Kool, Anne, Jakub Zavrel, and Walter Daelemans. 2000. Simultaneous feature selection and parameter optimization for memory-based natural language processing. In A. Feelders, editor, *BENELEARN 2000. Proceedings of the Tenth Belgian-Dutch Conference on Machine Learning*. Tilburg University, Tilburg, pages 93–100.

Krivanek, Julia and Detmar Meurers. 2011. Comparing rule-based and data-driven dependency parsing of learner language. In *Proceedings of the Int. Conference on Dependency Linguistics (Depling 2011)*.

Nilsson, Peter and Pierre Nugues. 2010. Automatic discovery of feature sets for dependency parsing. In *COLING*, pages 824–832.

Nivre, Joakim. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 149–160.

Nivre, Joakim. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34:513–553.

Nivre, Joakim. 2009. Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP (ACL-IJCNLP)*, pages 351–359.

Nivre, Joakim and Johan Hall. 2010. A quick guide to MaltParser optimization. Technical report, maltparser.org.

Nivre, Joakim, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task of EMNLP-CoNLL 2007*, pages 915–932.

Nivre, Joakim, Johan Hall, and Jens Nilsson. 2006. Maltparser: A data-driven parser-generator for dependency parsing. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC)*, pages 2216–2219.

Nivre, Joakim, Johan Hall, Jens Nilsson, G. Eryiğit, and S. Marinov. 2006. Labeled pseudo–projective dependency parsing with support vector machines. In *CoNLL–X*.

Nivre, Joakim, Marco Kuhlmann, and Johan Hall. 2009. An improved oracle for dependency parsing with online reordering. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 73–76.

Nivre, Joakim and Jens Nilsson. 2005. Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 99–106.

Ott, Niels and Ramon Ziai. 2010. Evaluating dependency parsing performance on German learner language. In *Proceedings of the Ninth International Workshop on Treebanks and Linguistic Theories (TLT9)*, NEALT Proceeding Series.

Prudhvi Kosaraju, Sruthilaya Reddy Kesidi, Vinay Bhargav Reddy Ainavolu and Puneeth Kukkadapu. 2010. Experiments on indian language dependency parsing. In *ICON-2010 Tools Contest on Indian Language Dependency Parsing. Kharagpur, India.*

Yli-Jyrä, Anssi Mikael. 2003. Multiplanarity – a model for dependency structures in treebanks. In Joakim Nivre and Erhard Hinrichs, editors, *TLT 2003. Proceedings of the Second Workshop on Treebanks and Linguistic Theories*, volume 9 of *Mathematical Modelling in Physics, Engineering and Cognitive Sciences*, pages 189–200, Växjö, Sweden, 14-15 November. Växjö University Press.